

Clique-Width and Directed Width Measures for Answer-Set Programming

Bernhard Bliem (bliem@dbai.tuwien.ac.at)

Sebastian Ordyniak (sordyniak@gmail.com)

Stefan Woltran (woltran@dbai.tuwien.ac.at)

TU Wien, Vienna, Austria

Abstract

Disjunctive Answer Set Programming (ASP) is a powerful declarative programming paradigm whose main decision problems are located on the second level of the polynomial hierarchy. Identifying tractable fragments and developing efficient algorithms for such fragments are thus important objectives in order to complement the sophisticated ASP systems available to date. Hard problems can become tractable if some problem parameter is bounded by a fixed constant; such problems are then called fixed-parameter tractable (FPT). While several FPT results for ASP exist, parameters that relate to directed or signed graphs representing the program at hand have been neglected so far. In this paper, we first give some negative observations showing that directed width measures on the dependency graph of a program do not lead to FPT results. We then consider the graph parameter of signed clique-width and present a novel dynamic programming algorithm that is FPT w.r.t. this parameter. Clique-width is more general than the well-known treewidth, and, to the best of our knowledge, ours is the first FPT algorithm for bounded clique-width for reasoning problems beyond SAT.

1 Introduction

Disjunctive Answer Set Programming (ASP) [10, 29, 44] is an active field of AI providing a declarative formalism for solving hard computational problems. Thanks to the high sophistication of modern solvers [28], ASP was successfully used in several applications, including product configuration [52], decision support for space shuttle flight controllers [2], team scheduling [49], and bio-informatics [33].

Since the main decision problems of propositional ASP are located at the second level of the polynomial hierarchy [24, 54], the quest for easier fragments are important research contributions that could lead to improvements in ASP systems. An interesting approach to dealing with intractable problems comes from parameterized complexity theory [22] and is based on the fact that many hard problems become polynomial-time tractable if some problem parameter is bounded by a fixed constant. If the order of the polynomial bound on the runtime is independent of the parameter, one speaks of *fixed-parameter tractability* (FPT). Results in this direction for the ASP domain include [43] (parameter: size of answer sets), [42] (number of cycles), [5] (length of longest cycles), [4] (number of non-Horn rules), and [26] (backdoors). Also related is the parameterized complexity analysis of reasoning under subset-minimal models, see, e.g., [41].

As many prominent representations of logic programs are given in terms of directed graphs (consider, e.g., the dependency graph), it is natural to investigate parameters for ASP that apply to directed graphs. Over the past two decades, various width measures for directed graphs have been introduced [3, 6, 35, 37, 50]. These are typically smaller than, e.g., the popular parameter of treewidth [7]. In particular, all these measures are zero on directed acyclic graphs (DAGs), but the treewidth of DAGs can be arbitrarily high. Moreover, since these measures are based on some notion of “closeness” to acyclicity and the complexity of ASP is closely related to the “cyclicity” of the rules in a program, such measures seem promising for obtaining efficient algorithms for ASP. Prominent applications of directed width measures include the k -Disjoint Path Problem [37], query evaluation in graph databases [1], and model checking [9].

Another graph parameter for capturing the structural complexity of a graph is clique-width [15–17]. It applies to directed and undirected graphs, and in its general form (known as signed clique-width) to edge-labeled graphs. It is defined via graph construction where only a limited number of vertex labels is available; vertices that share

the same label at a certain point of the construction process must be treated uniformly in subsequent steps. Constructions can be given by expressions in a graph grammar (so-called cwd-expressions) and the minimal number of labels required for constructing a graph G is the clique-width of G . While clique-width is in a certain way orthogonal to other directed width measures, it is more general than treewidth; there are classes of graphs with constant clique-width but arbitrarily high treewidth (e.g., complete graphs). In contrast, graphs with bounded treewidth also have bounded clique-width [12, 15].

By means of a meta-theorem due to Courcelle, Makowsky, and Rotics [18], one can solve any graph problem that can be expressed in Monadic Second-Order Logic with quantification on vertex sets (MSO_1) in linear time for graphs of bounded clique-width. This result is similar to Courcelle's theorem [13, 14] for graphs of bounded treewidth, which has been used for the FPT result for ASP w.r.t. treewidth [31]. There, the incidence graph of a program is used as an underlying graph structure (i.e., the graph containing a vertex for each atom a and rule r of the program, with an edge between a and r whenever a appears in r). Since the formula given in [31] is in MSO_1 , the FPT result for ASP applies also to signed clique-width.

Clique-width is NP-hard to compute [25], which might be considered as an obstacle toward practical applications. However, one can check in polynomial time whether the width of a graph is bounded by a fixed k [40, 47]. (These algorithms involve an additive approximation error that is bounded in terms of k). Recently, SAT solvers have been used to obtain sequences of vertex partitions that correspond to cwd-expressions [34] for a given graph. For some applications, it might not even be necessary to compute clique-width and the underlying cwd-expression: As mentioned in [27, Section 1.4], applications from the area of verification are supposed to already come with such an expression. Moreover, it might even be possible to partially obtain cwd-expressions during the grounding process of ASP.

This all calls for dedicated algorithms for solving ASP for programs of bounded clique-width. In contrast to treewidth where the FPT result from [31] has been used for designing [36] and implementing [45] a dynamic programming algorithm, to the best of our knowledge there are no algorithms yet that explicitly exploit the fixed-parameter tractability of ASP on bounded clique-width. In fact, we are not aware of any FPT algorithm for bounded clique-width for a reasoning problem located on the second level of the polynomial hierarchy (except [23] from the area of abstract argumentation).

The main contributions of this paper are as follows. First, we show some negative results for several *directed width measures*, indicating that the structure of the dependency graph and of various natural directed versions of the signed incidence graph does not adequately measure the complexity of evaluating the corresponding program.

Second, concerning *signed clique-width*, we give a novel dynamic programming algorithm that runs in polynomial time for programs where this parameter is bounded on their incidence graphs. We do so by suitably generalizing the seminal approach of [27] for the SAT problem. We also give a preliminary analysis how many signs are required in order to obtain FPT.

2 Preliminaries

Graphs. We use standard graph terminology, see for instance the handbook [21]. All our graphs are simple. An undirected graph G is a tuple (V, E) , where V or $V(G)$ is the vertex set and E or $E(G)$ is the edge set. For a subset $V' \subseteq V(G)$, we denote by $G[V']$, the *induced subgraph* of G induced by the vertices in V' , i.e., $G[V']$ has vertices V' and edges $\{\{u, v\} \in E(G) \mid u, v \in V'\}$. We also denote by $G \setminus V'$ the graph $G[V(G) \setminus V']$. Similarly to undirected graphs, a digraph D is a tuple (V, A) , where V or $V(D)$ is the vertex set and A or $A(D)$ is the *arc set*. A *strongly connected component* of a digraph D is a maximal subgraph Z of D that is strongly connected, i.e., Z contains a directed path between each pair of vertices in Z . We denote by $\text{UND}(D)$ the *symmetric closure* of D , i.e., the graph with vertex set $V(D)$ and arc set $\{(u, v), (v, u) \mid (u, v) \in A(D)\}$. Finally, for a directed graph D , we denote by $\text{DI}(G)$, the undirected graph with vertex set $V(G)$ and edge set $\{\{u, v\} \mid (u, v) \in A(D)\}$.

Parameterized Complexity. In parameterized algorithmics [22] the runtime of an algorithm is studied with respect to a parameter $k \in \mathbb{N}$ and input size n . The most favorable class is FPT (*fixed-parameter tractable*) which contains all problems that can be decided by an algorithm running in time $f(k) \cdot n^{\mathcal{O}(1)}$, where f is a computable function. We also call such an algorithm fixed-parameter tractable, or FPT for short. Formally, a

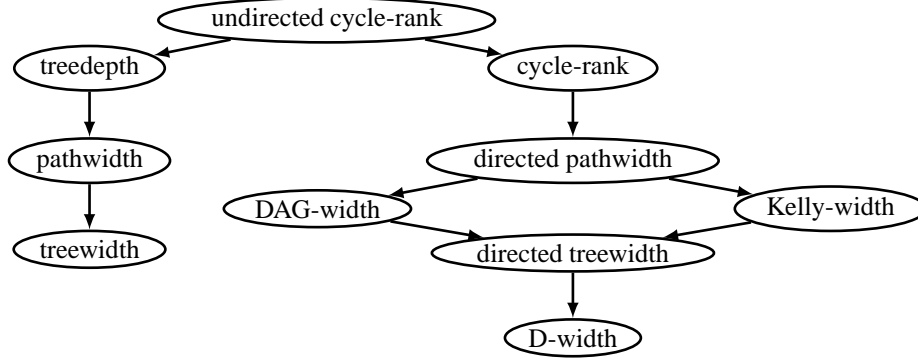


Figure 1: Propagation of hardness results for the considered width measures. An arc (A, B) indicates that any hardness result parameterized by measure A implies a corresponding hardness result parameterized by B .

parameterized problem is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is the input alphabet. Let $L_1 \subseteq \Sigma_1^* \times \mathbb{N}$ and $L_2 \subseteq \Sigma_2^* \times \mathbb{N}$ be two parameterized problems. A *parameterized reduction* (or FPT-reduction) from L_1 to L_2 is a mapping $P : \Sigma_1^* \times \mathbb{N} \rightarrow \Sigma_2^* \times \mathbb{N}$ such that: (1) $(x, k) \in L_1$ iff $P(x, k) \in L_2$, (2) the mapping can be computed by an FPT-algorithm w.r.t. parameter k , and (3) there is a computable function g such that $k' \leq g(k)$, where $(x', k') = P(x, k)$. The class $W[1]$ captures parameterized intractability and contains all problems that are FPT-reducible to PARTITIONED CLIQUE when parameterized by the size of the solution. Showing $W[1]$ -hardness for a problem rules out the existence of an FPT-algorithm under the usual assumption $FPT \neq W[1]$.

Answer Set Programming. A *program* Π consists of a set $\mathcal{A}(\Pi)$ of propositional atoms and a set $\mathcal{R}(\Pi)$ of rules of the form

$$a_1 \vee \dots \vee a_l \leftarrow a_{l+1}, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n,$$

where $n \geq m \geq l$ and $a_i \in \mathcal{A}(\Pi)$ for $1 \leq i \leq n$. Each rule $r \in \mathcal{R}(\Pi)$ consists of a head $h(r) = \{a_1, \dots, a_l\}$ and a body given by $p(r) = \{a_{l+1}, \dots, a_m\}$ and $n(r) = \{a_{m+1}, \dots, a_n\}$. A set $M \subseteq \mathcal{A}(\Pi)$ is called a *model* of r if $p(r) \subseteq M$ and $n(r) \cap M = \emptyset$ imply $h(r) \cap M \neq \emptyset$. We denote the set of models of r by $Mods(r)$ and the models of Π are given by $Mods(\Pi) = \bigcap_{r \in \mathcal{R}(\Pi)} Mods(r)$.

The *reduct* Π^I of a program Π with respect to a set of atoms $I \subseteq \mathcal{A}(\Pi)$ is the program Π^I with $\mathcal{A}(\Pi^I) = \mathcal{A}(\Pi)$ and $\mathcal{R}(\Pi^I) = \{r^+ \mid r \in \mathcal{R}(\Pi), n(r) \cap I = \emptyset\}$, where r^+ denotes rule r without negative body, i.e., $h(r^+) = h(r)$, $p(r^+) = p(r)$, and $n(r^+) = \emptyset$. Following [29], $M \subseteq \mathcal{A}(\Pi)$ is an *answer set* of Π if $M \in Mods(\Pi)$ and for no $N \subsetneq M$, we have $N \in Mods(\Pi^M)$. In what follows, we consider the problem of ASP consistency, i.e., the problem of deciding whether a given program has at least one answer set. As shown by Eiter and Gottlob, this problem is Σ_2^P -complete [24].

Graphical Representations of ASP. Let Π be a program. The *dependency graph* of Π , denoted by $DEP(\Pi)$, is the directed graph with vertex set $\mathcal{A}(\Pi)$ and that contains an arc (x, y) if there is a rule $r \in \mathcal{R}(\Pi)$ such that either $x \in h(r)$ and $y \in p(r) \cup n(r)$ or $x, y \in h(r)$ [26]. Note that there are other notions of dependency graphs used in the literature, most of them, however, are given as subgraphs of $DEP(\Pi)$. As we will see later, our definition of dependency graphs allows us to draw immediate conclusions for such other notions.

The *incidence graph* of Π , denoted by $INC(\Pi)$, is the undirected graph with vertices $\mathcal{A}(\Pi) \cup \mathcal{R}(\Pi)$ that contains an edge between a *rule vertex* $r \in \mathcal{R}(\Pi)$ and a *atom vertex* $a \in \mathcal{A}(\Pi)$ whenever $a \in h(r) \cup p(r) \cup n(r)$. The *signed incidence graph* of Π , denoted by $SINC(\Pi)$, is the graph $INC(\Pi)$, where additionally every edge of $INC(\Pi)$ between an atom a and a rule r is annotated with a label from $\{h, p, n\}$ depending on whether a occurs in $h(r)$, $p(r)$, or $n(r)$.

3 Directed Width Measures

Since many representations of ASP programs are in terms of directed graphs, it is natural to consider parameters for ASP that are tailor-made for directed graphs. Over the past two decades various width measures for directed graphs have been introduced, which are better suited for directed graphs than treewidth, on which they are based. The most prominent of those are directed treewidth [37], directed pathwidth [3], DAG-width [6], Kelly-width [35], and D-width [50] (see also [20]). Since these width measures are usually smaller on directed graphs than treewidth, it is worth considering them for problems that have already been shown to be fixed-parameter tractable parameterized by treewidth. In particular, all of these measures are zero on directed acyclic graphs (DAGs), but the treewidth of DAGs can be arbitrary high. Moreover, since these measures are based on some notion of “closeness” to acyclicity and the complexity of ASP is closely related to the “cyclicity” of the logical rules, one would consider such measures as promising for obtaining efficient algorithms for ASP.

In this section, we give results for directed width measures when applied to dependency graphs as defined in Section 2. To state our results in the most general manner, we will employ the parameter cycle-rank [11]. Since the cycle-rank is always greater or equal to any of the above mentioned directed width measures [32, 38], any (parameterized) hardness result obtained for cycle-rank carries over to the aforementioned width measures for directed graphs.

Definition 1. Let $D = (V, A)$ be a directed graph. The cycle-rank of D , denoted by $\text{cr}(D)$, is inductively defined as follows: if D is acyclic, then $\text{cr}(D) = 0$. Moreover, if D is strongly connected, then $\text{cr}(D) = 1 + \min_{v \in V} \text{cr}(D \setminus \{v\})$. Otherwise the cycle-rank of D is the maximum cycle-rank of any strongly connected component of D .

We will also consider a natural “undirected version” of the cycle-rank for directed graphs, i.e., we define the *undirected cycle-rank* of a directed graph D , denoted by $\text{cr}^{\leftrightarrow}(D)$, to be the cycle-rank of $\text{UND}(D)$. It is also well known (see, e.g., [30]) that the cycle-rank of $\text{UND}(D)$ is equal to the treedepth of $\text{DI}(D)$, i.e., the underlying undirected graph of D , and that the treedepth is always an upper bound for the pathwidth and the treewidth of an undirected graph [8]. Putting these facts together implies that any hardness result obtained for the undirected cycle-rank implies hardness for pathwidth, treewidth, treedepth as well as the aforementioned directed width measures. See also Figure 1 for an illustration how hardness results for the considered width measures propagate.

Finally, we would like to remark that both the cycle-rank and the undirected cycle-rank are easily seen to be closed under taking subgraphs, i.e., the (undirected) cycle-rank of a graph is always larger or equal to the (undirected) cycle-rank of every subgraph of the graph.

Hardness Results

We show that ASP consistency remains as hard as in the general setting even for instances that have a dependency graph of constant width in terms of any of the directed width measures introduced.

For our hardness results, we employ the reduction given in [24] showing that ASP consistency is Σ_2^P -hard in general. The reduction is given from the validity problem for quantified Boolean formulas (QBF) of the form: $\Phi := \exists x_1 \dots \exists x_n \forall y_1 \dots \forall y_m \bigvee_{j=1}^r D_j$ where each D_j is a conjunction of at most three literals over the variables x_1, \dots, x_n and y_1, \dots, y_m . We will denote the set of all QBF formulas of the above form in the following by $\text{QBF}_{2,\exists}^{\text{DNF}}$.

Given $\Phi \in \text{QBF}_{2,\exists}^{\text{DNF}}$, a program $\Pi(\Phi)$ is constructed as follows. The atoms of $\Pi(\Phi)$ are $x_1, v_1, \dots, x_n, v_n, y_1, z_1, \dots, y_m, z_m$, and w and $\Pi(\Phi)$ contains the following rules:

- for every i with $1 \leq i \leq n$, the rule $x_i \vee v_i \leftarrow$,
- for every i with $1 \leq i \leq m$, the rules $y_i \vee z_i \leftarrow$, $y_i \leftarrow w$, $z_i \leftarrow w$, and $w \leftarrow y_i, z_i$,
- for every j with $1 \leq j \leq r$, the rule $w \leftarrow \sigma(L_{j,1}), \sigma(L_{j,2}), \sigma(L_{j,3})$, where $L_{j,l}$ (for $l \in \{1, 2, 3\}$) is the l -th literal that occurs in D_j (if $|D_j| < 3$, the respective parts are omitted) and the function σ is defined by setting $\sigma(L)$ to v_i if $L = \neg x_i$, to z_i if $L = \neg y_i$, and to L otherwise.
- the rule $\leftarrow \neg w$ (i.e., with an empty disjunction in the head).

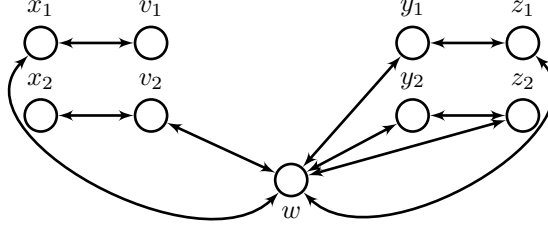


Figure 2: The symmetric closure of the dependency graph of the program $\Pi(\Phi)$ for the formula $\Phi := \exists x_1 \exists x_2 \forall y_1 \forall y_2 (x_1 \wedge \neg y_2) \vee (\neg x_2 \wedge y_2)$. Here $\Pi(\Phi)$ contains the rules $x_i \vee v_i \leftarrow$, $y_i \vee z_i \leftarrow$, $y_i \leftarrow w$, $z_i \leftarrow w$, for every $i \in \{1, 2\}$ and the rules $w \leftarrow x_1, z_2$, $w \leftarrow v_2, y_2$.

It has been shown [24, Theorem 38] that a $\text{QBF}_{2,\exists}^{\text{DNF}}$ formula Φ is valid iff $\Pi(\Phi)$ has an answer set. As checking validity of $\text{QBF}_{2,\exists}^{\text{DNF}}$ formulas is Σ_2^P -complete [53], this reduction shows that ASP is Σ_2^P -hard.

Lemma 1. *Let Φ be a $\text{QBF}_{2,\exists}^{\text{DNF}}$, then $\text{cr}^{\leftrightarrow}(\text{DEP}(\Pi(\Phi))) \leq 2$.*

Proof. Figure 2 illustrates the symmetric closure of $\text{DEP}(\Pi(\Phi))$ for a simple $\text{QBF}_{2,\exists}^{\text{DNF}}$ formula Φ . As this example illustrates, the only arcs in $\text{UND}(\text{DEP}(\Pi(\Phi)))$ not incident to w are the arcs incident to x_i and v_i and the arcs incident to y_j and z_j , for $1 \leq i \leq n$ and $1 \leq j \leq m$. Hence, after removing w from $\text{UND}(\text{DEP}(\Pi(\Phi)))$, every strongly connected component of the remaining graph contains at most two vertices and each of those has hence cycle-rank at most one. It follows that the cycle-rank of $\text{UND}(\text{DEP}(\Pi(\Phi)))$ and hence the undirected cycle-rank of $\text{DEP}(\Pi(\Phi))$ is at most two. \square

Together with our considerations from above, we obtain:

Theorem 1. *ASP consistency is Σ_2^P -complete even for instances whose dependency graph has width at most two for any of the following width measures: undirected cycle-rank, pathwidth, treewidth, treedepth, cycle-rank, directed treewidth, directed pathwidth, DAG-width, Kelly-width, and D-width.*

Observe that because the undirected cycle-rank is closed under taking subgraphs and we chose the “richest” variant of the dependency graph, the above result carries over to the other notions of dependency graphs of ASP programs considered in the literature.

The above result draws a very negative picture of the complexity of ASP w.r.t. restrictions on the dependency graph. In particular, not even structural restrictions of the dependency graph by the usually very successful parameter treewidth can be employed for ASP. This is in contrast to our second graphical representation of ASP, the incidence graph, for which it is known that ASP is fixed-parameter tractable parameterized by the treewidth [36]. It is hence natural to ask whether the same still holds under restrictions provided by one of the directed width measures under consideration. We first need to discuss how to obtain a directed version of the usually undirected incidence graph. For this, observe that the incidence graph, unlike the signed incidence graph, provides merely an incomplete model of the underlying ASP instance. Namely, it misses the information about *how* atoms occur in rules, i.e., whether they occur in the head, in the positive body, or in the negative body of a rule. A directed version of the incidence graph should therefore use the additional expressiveness provided by the direction of the arcs to incorporate the information given by the labels of the signed incidence graph. For instance, a natural directed version of the incidence graph could orient the edges depending on whether an atom occurs in the head or in the body of a rule. Clearly, there are many ways to orient the edges and it is not a priori clear which of those orientations leads to a directed version of the incidence graph that is best suited for an application of the directed width measures. Every orientation should, however, be consistent with the labels of the signed incidence graph, i.e., whenever two atoms are connected to a rule via edges having the same label, their arcs should be oriented in the same way. We call such an orientation of the incidence graph a *homogeneous orientation*.

Lemma 2. *Let Φ be a $\text{QBF}_{2,\exists}^{\text{DNF}}$, then the cycle-rank of any homogeneous orientation of the incidence graph of $\Pi(\Phi)$ is at most one.*

Proof. Let D be a homogeneous orientation of $\text{INC}(\Pi(\Phi))$ and let $G = \text{SINC}(\Pi(\Phi))$. First observe that in $G \setminus \{w\}$ every rule vertex is either only incident to edges with label h or to edges of label p . Hence, as D is a homogeneous orientation, we obtain that every rule vertex of $D \setminus \{w\}$ is either a source vertex (i.e., having only outgoing arcs) or a sink vertex (i.e., having only incoming arcs). So $D \setminus \{w\}$ cannot contain a cycle through a rule vertex. However, since there are no arcs between atom vertices in D , we obtain that $D \setminus \{w\}$ is acyclic, which shows that the cycle-rank of D is at most one. \square

We can thus state the following result:

Theorem 2. *ASP consistency is Σ_2^P -complete even for instances whose directed incidence graph has width at most one for any of the following width measures: cycle-rank, directed treewidth, directed pathwidth, DAG-width, Kelly-width, and D-width.*

4 Clique-Width

The results in [31] imply that bounding the clique-width of the signed incidence graph of a program leads to tractability.

Proposition 1. *For a program Π such that the clique-width of its signed incidence graph is bounded by a constant, we can decide in linear time whether Π has an answer set.*

This result has been established via a formulation of ASP consistency as an MSO_1 formula. Formulating a problem in this logic automatically gives us an FPT algorithm. However, such algorithms are primarily of theoretical interest due to huge constant factors, and for actually solving problems, it is preferable to explicitly design dynamic programming algorithms [19].

Since our main tractability result considers the clique-width of an edge-labeled graph, i.e., the signed incidence graph, we will introduce clique-width for edge-labeled graphs. This definition also applies to graphs without edge-labels by considering all edges to be labeled with the same label. A k -graph, for $k > 0$, is a graph whose vertices are labeled by integers from $\{1, \dots, k\} =: [k]$. Additionally, we also allow for the edges of a k -graph to be labeled by some arbitrary but finite set of labels (in our case the labels will correspond to the signs of the signed incidence graph). The labeling of the vertices of a graph $G = (V, E)$ is formally denoted by a function $\mathcal{L} : V \rightarrow [k]$. We consider an arbitrary graph as a k -graph with all vertices labeled by 1. We call the k -graph consisting of exactly one vertex v (say, labeled by $i \in [k]$) an *initial k -graph* and denote it by $i(v)$.

Graphs can be constructed from initial k -graphs by means of repeated application of the following three operations.

- *Disjoint union* (denoted by \oplus);
- *Relabeling*: changing all labels i to j (denoted by $\rho_{i \rightarrow j}$);
- *Edge insertion*: connecting all vertices labeled by i with all vertices labeled by j via an edge with label ℓ (denoted by $\eta_{i,j}^\ell$); $i \neq j$; already existing edges are not doubled.

A construction of a k -graph G using the above operations can be represented by an algebraic term composed of $i(v)$, \oplus , $\rho_{i \rightarrow j}$, and $\eta_{i,j}^\ell$, ($i, j \in [k]$, and v a vertex). Such a term is then called a *cwd-expression defining G* . For any cwd-expression σ , we use $\mathcal{L}_\sigma : V \rightarrow [k]$ to denote the labeling of the graph defined by σ . A k -expression is a cwd-expression in which at most k different labels occur. The set of all k -expressions is denoted by CW_k .

As an example consider the complete bipartite graph $K_{n,n}$ with bipartition $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$ and assume that all edges of $K_{n,n}$ are labeled with the label ℓ . A cwd-expression of $K_{n,n}$ using at most two labels is given by the following steps: (1) introduce all vertices in A using label 1, (2) introduce all vertices in B using label 2, (3) take the disjoint union of all these vertices, and (4) add all edges between vertices with label 1 and vertices with label 2, i.e., such a cwd-expression is given by $\eta_{1,2}^\ell(1(a_1) \oplus \dots \oplus 1(a_n) \oplus 2(b_1) \oplus \dots \oplus 2(b_n))$. As a second example consider the complete graph K_n on n vertices, where all edges are labeled with label ℓ . A cwd-expression for K_n using at most two labels can be obtained by the following iterative process: Given a cwd-expression σ_{n-1} for K_{n-1} , where every vertex is labeled with label 1, one takes the disjoint union

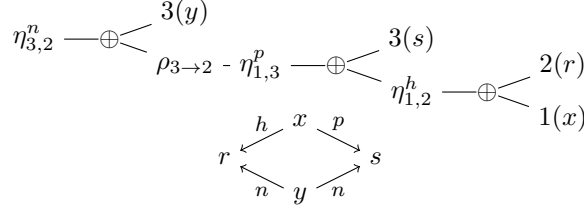


Figure 3: A parse tree (top) of a 3-expression for $\text{SINC}(\Pi)$ (bottom), where Π is the program consisting of the rules $x \leftarrow \neg y$ and $\leftarrow x, \neg y$

of σ_{n-1} and $2(v)$ (where v is the vertex only contained in K_n but not in K_{n-1}), adds all edges between vertices with label 1 and vertices with label 2, and then relabels label 2 to label 1. Formally, the cwd-expression σ_n for K_n is given by $(\rho_{2 \rightarrow 1}(\eta_{1,2}^\ell(\sigma_{n-1} \oplus 2(v_2))))$.

Definition 2. The clique-width of a graph G , $\text{cwd}(G)$, is the smallest integer k such that G can be defined by a k -expression.

Our discussion above thus witnesses that complete (bipartite) graphs have clique-width 2. Furthermore, co-graphs also have clique-width 2 (co-graphs are exactly given by the graphs which are P_4 -free, i.e., whenever there is a path (a, b, c, d) in the graph then $\{a, c\}$, $\{a, d\}$ or $\{b, d\}$ is also an edge of the graph) and trees have clique-width 3.

We have already introduced the notion of incidence graphs (resp. signed incidence graphs) of a program in Section 2. We thus can use cwd-expressions to represent programs.

Example 1. Let Π be the program with $\mathcal{A}(\Pi) = \{x, y\}$ and $\mathcal{R}(\Pi) = \{r, s\}$, where r is the rule $x \leftarrow \neg y$ and s is the rule $\leftarrow x, \neg y$. Its signed incidence graph $\text{SINC}(\Pi)$ can be constructed by the 3-expression $\eta_{3,2}^n(\rho_{3 \rightarrow 2}(\eta_{1,3}^p(\eta_{1,2}^h(1(x) \oplus 2(r)) \oplus 3(s))) \oplus 3(y))$, as depicted in Figure 3.

Since every k -expression of the signed incidence graph can be transformed into a k -expression of the unsigned incidence graph (by replacing all operations of the form $\eta_{i,j}^\ell$ with $\eta_{i,j}^\alpha$, where α is new label), it holds that $\text{cwd}(\text{INC}(\Pi)) \leq \text{cwd}(\text{SINC}(\Pi))$.

Proposition 2. Let Π be a program. It holds that $\text{cwd}(\text{INC}(\Pi)) \leq \text{cwd}(\text{SINC}(\Pi))$, and there is a class \mathcal{C} of programs such that, for each $\Pi \in \mathcal{C}$, $\text{cwd}(\text{INC}(\Pi)) = 2$ but $\text{cwd}(\text{SINC}(\Pi))$ is unbounded.

For showing the second statement of the above proposition, consider a program Π_n that has n^2 atoms and n^2 rules (for some $n \in \mathbb{N}$), such that every atom occurs in every rule of Π_n . Because the incidence graph is a complete bipartite graph it has clique-width two and moreover it contains a grid G of size $n \times n$ as a subgraph. Assume that Π_n is defined in such a way that an atom a occurring in a rule r is in the head of r if the edge between a and r occurs in the grid G and otherwise a is in the (positive) body of r . Then, the clique-width of $\text{SINC}(\Pi_n)$ is at least the clique-width of the $n \times n$ grid G , which grows with n [39]. Hence, the class \mathcal{C} containing Π_n for every $n \in \mathbb{N}$ shows the second statement of the above proposition.

4.1 Algorithms

In this section, we provide our dynamic programming algorithms for deciding existence of an answer set. We start with the classical semantics for programs, where it is sufficient to just slightly adapt (a simplified version of) the algorithm for SAT by [27]. For answer-set semantics, we then extend this algorithm in order to deal with the intrinsic higher complexity of this semantics.

Both algorithms follow the same basic principles by making use of a k -expression σ defining a program Π via its signed incidence graph in the following way: We assign certain objects to each subexpression of σ and manipulate these objects in a bottom-up traversal of the parse tree of the k -expression such that the objects in the root of the parse tree then provide the necessary information to decide the problem under consideration. The size of these objects is bounded in terms of k (and independent of the size of Π) and the number of such objects required is linear in the size of Π . Most importantly, we will show that these objects can also be efficiently computed for bounded k . Thus, we will obtain the desired linear running time.

4.1.1 Classical Semantics

Definition 3. A tuple $Q = (T, F, U)$ with $T, F, U \subseteq [k]$ is called a k -triple, and we refer to its parts using $Q_T = T$, $Q_F = F$, and $Q_U = U$. The set of all k -triples is given by \mathcal{Q}_k .

The intuition of a triple (T, F, U) is to characterize a set of interpretations I in the following way:

- For each $i \in T$, at least one atom with label i is true in I ;
- for each $i \in F$, at least one atom with label i is false in I ;
- for each $i \in U$, there is at least one rule with label i that is “not satisfied yet”.

Formally, the “semantics” of a k -triple Q with respect to a given program Π is given as follows.

Definition 4. Let $Q \in \mathcal{Q}_k$ and Π be a program whose signed incidence graph (V, E) is labeled by $\mathcal{L} : V \rightarrow [k]$. A Π -interpretation of Q is a set $I \subseteq \mathcal{A}(\Pi)$ that satisfies

$$\begin{aligned} Q_T &= \{\mathcal{L}(a) \mid a \in I\}, \\ Q_F &= \{\mathcal{L}(a) \mid a \in \mathcal{A}(\Pi) \setminus I\}, \text{ and} \\ Q_U &= \{\mathcal{L}(r) \mid r \in \mathcal{R}(\Pi), I \notin \text{Mods}(r)\}. \end{aligned}$$

Example 2. Consider again program Π from Example 1 and the 3-expression σ from Figure 3. Let Q be the 3-triple $(\{1\}, \{3\}, \{2\})$. Observe that $\{x\}$ is a Π -interpretation of Q : It sets x to true and y to false, and $\mathcal{L}_\sigma(x) \in Q_T$ and $\mathcal{L}_\sigma(y) \in Q_F$ hold as required; the rule s is not satisfied by $\{x\}$, and indeed $\mathcal{L}_\sigma(s) \in Q_U$. We can easily verify that no other subset of $\mathcal{A}(\Pi)$ is a Π -interpretation of Q : Each Π -interpretation of Q must set x to true and y to false, as these are the only atoms labeled with 1 and 3, respectively.

We use the following notation for k -triples Q, Q' , and set $S \subseteq [k]$.

- $Q \oplus Q' = (Q_T \cup Q'_T, Q_F \cup Q'_F, Q_U \cup Q'_U)$
- $Q^{i \rightarrow j} = (Q_T^{i \rightarrow j}, Q_F^{i \rightarrow j}, Q_U^{i \rightarrow j})$ where for $S \subseteq [k]$,

$$S^{i \rightarrow j} = S \setminus \{i\} \cup \{j\} \text{ if } i \in S \text{ and } S^{i \rightarrow j} = S \text{ otherwise.}$$

- $Q^{S, i, j} = (Q_T, Q_F, Q_U \setminus \{j\})$ if $i \in S$; $Q^{S, i, j} = Q$ otherwise.

Using these abbreviations, we define our dynamic programming algorithm: We assign to each subexpression σ of a given k -expression a set of triples by recursively defining a function f , which associates to σ a set of k -triples as follows.

Definition 5. The function $f : CW_k \rightarrow 2^{\mathcal{Q}_k}$ is recursively defined along the structure of k -expressions as follows.

- $f(i(v)) = \begin{cases} \{(\{i\}, \emptyset, \emptyset), (\emptyset, \{i\}, \emptyset)\} & \text{if } v \text{ is an atom node} \\ \{(\emptyset, \emptyset, \{i\})\} & \text{if } v \text{ is a rule node} \end{cases}$
- $f(\sigma_1 \oplus \sigma_2) = \{Q \oplus Q' \mid Q \in f(\sigma_1), Q' \in f(\sigma_2)\}$
- $f(\rho_{i \rightarrow j}(\sigma)) = \{Q^{i \rightarrow j} \mid Q \in f(\sigma)\}$
- $f(\eta_{i,j}^h(\sigma)) = f(\eta_{i,j}^n(\sigma)) = \{Q^{Q_T, i, j} \mid Q \in f(\sigma)\}$
- $f(\eta_{i,j}^p(\sigma)) = \{Q^{Q_F, i, j} \mid Q \in f(\sigma)\}$

Example 3. Consider again program Π from Example 1 and the 3-expression depicted in Figure 3. To break down the structure of σ , let $\sigma_1, \dots, \sigma_6$ be subexpressions of σ such that $\sigma = \eta_{3,2}^n(\sigma_1)$, $\sigma_1 = \sigma_2 \oplus 3(y)$, $\sigma_2 = \rho_{3 \rightarrow 2}(\sigma_3)$, $\sigma_3 = \eta_{1,3}^p(\sigma_4)$, $\sigma_4 = \sigma_5 \oplus 3(s)$, $\sigma_5 = \eta_{1,2}^h(\sigma_6)$ and $\sigma_6 = 1(x) \oplus 2(r)$. We get $f(1(x)) = \{(\{1\}, \emptyset, \emptyset), (\emptyset, \{1\}, \emptyset)\}$ and $f(2(r)) = \{(\emptyset, \emptyset, \{2\})\}$. These sets are then combined to $f(\sigma_6) = \{(\{1\}, \emptyset, \{2\}), (\emptyset, \{1\}, \{2\})\}$. The program defined by σ_6 consists of atom x and rule r , but x does not occur in r yet. Accordingly, the k -triple $(\{1\}, \emptyset, \{2\})$ models the situation where x is set to true, which does not satisfy r (since the head and body of r are still empty), hence the label of r is in the last component; the 3-triple $(\emptyset, \{1\}, \{2\})$ represents x being set to false, which does not satisfy r either. Next, σ_5 causes all atoms with label 1 (i.e., just x) to be inserted into the head of all rules with label 2 (i.e., just r), and we get $f(\sigma_5) = \{(\{1\}, \emptyset, \emptyset), (\emptyset, \{1\}, \{2\})\}$. We obtain the first element $(\{1\}, \emptyset, \emptyset) = Q_{T,1,2}^{Q_U}$ from $Q = (\{1\}, \emptyset, \{2\})$ by removing the label 2 from Q_U because $1 \in Q_T$. The idea is that the heads of all rules labeled with 2 now contain all atoms labeled with 1, so these rules become satisfied by every interpretation that sets some atom labeled with 1 to true. Next, σ_4 adds the rule s with label 3 and we get $f(\sigma_4) = \{(\{1\}, \emptyset, \{3\}), (\emptyset, \{1\}, \{2, 3\})\}$. The edge added by σ_3 adds all atoms with label 1 (i.e., just x) into the positive body of all rules with label 3 (i.e., just s), which results in $f(\sigma_3) = \{(\{1\}, \emptyset, \{3\}), (\emptyset, \{1\}, \{2\})\}$. Observe that the last component of the second element no longer contains 3, i.e., setting x to false makes s true. Now the label 3 is renamed to 2, and we get $f(\sigma_2) = \{(\{1\}, \emptyset, \{2\}), (\emptyset, \{1\}, \{2\})\}$. Note that now r and s are no longer distinguishable since they now share the same label. Hence all operations that add edges to r will also add edges to s and vice versa. In σ_1 , atom y is added with label 3 and we get four 3-triples in $f(\sigma_1)$: From $(\{1\}, \emptyset, \{2\})$ in $f(\sigma_2)$ we obtain $(\{1, 3\}, \emptyset, \{2\})$ and $(\{1\}, \{3\}, \{2\})$, and from $(\emptyset, \{1\}, \{2\})$ in $f(\sigma_2)$ we get $(\{3\}, \{1\}, \{2\})$ and $(\emptyset, \{1, 3\}, \{2\})$. In σ , we add a negative edge from all atoms labeled with 3 (i.e., just y) to all rules labeled with 2 (both r and s). From $(\{1, 3\}, \emptyset, \{2\})$ in $f(\sigma_1)$ we now get $(\{1, 3\}, \emptyset, \emptyset)$, from $(\{3\}, \{1\}, \{2\})$ we get $(\{3\}, \{1\}, \emptyset)$, and the 3-triples $(\{1\}, \{3\}, \{2\})$ and $(\emptyset, \{1, 3\}, \{2\})$ from $f(\sigma_1)$ occur unmodified in $f(\sigma)$. As we will prove shortly, for each k -triple Q in $f(\sigma)$, there is a Π -interpretation of Q . So if there is a k -triple Q in $f(\sigma)$ such that $Q_U = \emptyset$, then Π has a classical model due to the definition of Q_U . For instance, $(\{1, 3\}, \emptyset, \emptyset)$ has a Π -interpretation $\{x, y\}$, which is obviously a model of Π .

We now prove correctness of our algorithm:

Lemma 3. Let Π be a program and θ be a k -expression for $\text{SINC}(\Pi)$. For every set $I \subseteq \mathcal{A}(\Pi)$, there is a k -triple $Q \in f(\theta)$ such that I is a Π -interpretation of Q , and for every k -triple $Q \in f(\theta)$ there is a set $I \subseteq \mathcal{A}(\Pi)$ such that I is a Π -interpretation of Q .

Proof. We prove the first statement by induction on the structure of a k -expression θ defining Π . Let σ be a subexpression of θ , let Π_σ denote the program defined by σ , and let $I \subseteq \mathcal{A}(\Pi_\sigma)$.

If $\sigma = i(r)$, for $r \in \mathcal{R}(\Pi)$, then $\mathcal{A}(\Pi_\sigma) = \emptyset$, so $I = \emptyset$. Moreover, $\mathcal{R}(\Pi_\sigma)$ consists of an unsatisfiable rule (its head and body are empty). Hence I is a Π_σ -interpretation of $(\emptyset, \emptyset, \{i\})$ in $f(\sigma)$.

If $\sigma = i(a)$, for $a \in \mathcal{A}(\Pi)$, then $\mathcal{A}(\Pi_\sigma) = \{a\}$ and $\mathcal{R}(\Pi_\sigma) = \emptyset$. If $I = \emptyset$, then I is a Π_σ -interpretation of the k -triple $(\emptyset, \{i\}, \emptyset)$ in $f(\sigma)$. Otherwise $I = \{a\}$ and I is a Π_σ -interpretation of the k -triple $(\{i\}, \emptyset, \emptyset)$ in $f(\sigma)$.

If $\sigma = \sigma_1 \oplus \sigma_2$, let $i \in \{1, 2\}$, $\Pi_i = \Pi_{\sigma_i}$ and $I_i = I \cap \mathcal{A}(\Pi_i)$. By definition of Π_i , it holds that $\mathcal{A}(\Pi) = \mathcal{A}(\Pi_1) \cup \mathcal{A}(\Pi_2)$, $\mathcal{R}(\Pi) = \mathcal{R}(\Pi_1) \cup \mathcal{R}(\Pi_2)$ and $I = I_1 \cup I_2$. By induction hypothesis, I_i is a Π_i -interpretation of some k -triple Q_i in $f(\sigma_i)$. By definition of f , there is a k -triple Q in $f(\sigma)$ with $Q_T = Q_{1T} \cup Q_{2T}$, $Q_F = Q_{1F} \cup Q_{2F}$ and $Q_U = Q_{1U} \cup Q_{2U}$. This allows us to easily verify that I is a Π_σ -interpretation of Q by checking the conditions in Definition 4.

If $\sigma = \rho_{i \rightarrow j}(\sigma')$, then $\Pi_\sigma = \Pi_{\sigma'}$ and, by induction hypothesis, I is a $\Pi_{\sigma'}$ -interpretation of some k -triple Q in $f(\sigma')$. By definition of f , the k -triple $Q^{i \rightarrow j}$ in $f(\sigma)$ is the result of replacing i by j in each of Q_T , Q_F and Q_U . Hence we can easily verify that I satisfies all conditions for being a Π_σ -interpretation of $Q^{i \rightarrow j}$.

If $\sigma = \eta_{i,j}^\ell(\sigma')$, for $\ell \in \{h, n\}$, then $\mathcal{A}(\Pi_\sigma) = \mathcal{A}(\Pi_{\sigma'})$. Hence, by induction hypothesis, I is a $\Pi_{\sigma'}$ -interpretation of some k -triple Q' in $f(\sigma')$. We use Q to denote the k -triple $Q'^{Q'_T, i, j}$, which is in $f(\sigma)$. Since $Q_T = Q'_T$, $Q_F = Q'_F$ and $\mathcal{L}_\sigma = \mathcal{L}_{\sigma'}$, I satisfies the first two conditions for being a Π_σ -interpretation of Q . It remains to check the third condition.

For every $j' \in [k] \setminus \{j\}$ it holds that $j' \in Q_U$ if and only if $j' \in Q'_U$. By induction hypothesis, the latter is the case if and only if there is a rule $r' \in \mathcal{R}(\Pi_{\sigma'})$ such that $\mathcal{L}_{\sigma'}(r') = j'$ and $I \notin \text{Mods}(r')$. This is equivalent

to the existence of a rule $r \in \mathcal{R}(\Pi_\sigma)$ such that $\mathcal{L}_\sigma(r) = j'$, $h(r) = h(r')$, $p(r) = p(r')$, $n(r) = n(r')$ and $I \notin \text{Mods}(r)$, since $\text{SINC}(\Pi_\sigma)$ only differs from $\text{SINC}(\Pi_{\sigma'})$ by additional edges that are not incident to r due to $j' \neq j$.

It remains to check that $j \in Q_U$ if and only if there is a rule $r \in \mathcal{R}(\Pi_\sigma)$ such that $\mathcal{L}_\sigma(r) = j$ and $I \notin \text{Mods}(r)$. First suppose toward a contradiction that $j \in Q_U$ while I is a model of every rule $r \in \mathcal{R}(\Pi_\sigma)$ such that $\mathcal{L}_\sigma(r) = j$. Since $Q_U \subseteq Q'_U$, also $j \in Q'_U$ and by induction hypothesis there is a rule $r' \in \mathcal{R}(\Pi_{\sigma'})$ such that $\mathcal{L}_{\sigma'}(r') = j$ and I is not a model of r' . There is a corresponding rule $r \in \mathcal{R}(\Pi_\sigma)$, for which $\mathcal{L}_\sigma(r) = j$, $h(r') \subseteq h(r)$, $n(r') \subseteq n(r)$ and $p(r') = p(r)$ hold. Since I is a model of r but not of r' , I contains some atom labeled with i (by both $\mathcal{L}_{\sigma'}$ and \mathcal{L}_σ) because all atoms in $h(r) \setminus h(r')$ and $n(r) \setminus n(r')$ are labeled with i . By induction hypothesis, this implies $i \in Q'_T$, which leads to the contradiction $j \notin Q_U$ by construction of f .

Finally, suppose toward a contradiction that $j \notin Q_U$ and there is a rule $r \in \mathcal{R}(\Pi_\sigma)$ such that $\mathcal{L}_\sigma(r) = j$ and $I \notin \text{Mods}(r)$. The rule r' corresponding to r in $\Pi_{\sigma'}$ with $\mathcal{L}_{\sigma'}(r') = j$ is not satisfied by I either, since $h(r') \subseteq h(r)$, $n(r') \subseteq n(r)$ and $p(r') = p(r)$. By induction hypothesis, this entails $j \in Q'_U$. Due to $j \notin Q_U$, it holds that $i \in Q'_T$, so there is an $a \in I$ with $\mathcal{L}_{\sigma'}(a) = \mathcal{L}_\sigma(a) = i$. Due to the new edge from a to r , either $a \in h(r)$ or $a \in n(r)$. This yields the contradiction that I is a model of r .

The case $\sigma = \eta_{i,j}^p(\sigma')$ is symmetric.

The proof of the second statement is similar. \square

We can now state our FPT result for classical models:

Theorem 3. *Let k be an integer and Π be a program. Given a k -expression for the signed incidence graph of Π , we can decide in linear time whether Π has a model.*

Proof. Let k be a constant, Π be a program and σ be a k -expression of $\text{SINC}(\Pi)$. We show that there is a model of Π if and only if there is a k -triple Q in $f(\sigma)$ with $Q_U = \emptyset$: If Π has a model I , then I is a Π -interpretation of a k -triple Q in $f(\sigma)$, by Lemma 3, and $Q_U = \emptyset$ by Definition 4. Conversely, if there is a k -triple Q in $f(\sigma)$ with $Q_U = \emptyset$, then there is a Π -interpretation I of Q , by Lemma 3, and $Q_U = \emptyset$ implies that I is a model of Π by Definition 4. Finally, it is easy to see that $f(\sigma)$ can be computed in linear time. \square

4.1.2 Answer-Set Semantics

For full disjunctive ASP we need a more involved data structure.

Definition 6. A pair (Q, Γ) with $Q \in \mathcal{Q}_k$ and $\Gamma \subseteq \mathcal{Q}_k$ is called a k -pair. The set of all k -pairs is given by \mathcal{P}_k .

Given a k -pair (Q, Γ) , the purpose of Q is, as for classical semantics, to represent Π -interpretations I (that in the end correspond to models). Every k -triple in Γ represents sets J of atoms such that $J \subset I$. If, in the end, there is such a set J that still satisfies every rule in the reduct w.r.t. I , then we conclude that I is not an answer set.

Definition 7. Let $Q \in \mathcal{Q}_k$, let Π be a program whose signed incidence graph (V, E) is labeled by $\mathcal{L} : V \rightarrow [k]$, and let $I \subseteq \mathcal{A}(\Pi)$. A Π^I -interpretation of Q is a set $J \subseteq \mathcal{A}(\Pi)$ such that

$$\begin{aligned} Q_T &= \{\mathcal{L}(a) \mid a \in J\}, \\ Q_F &= \{\mathcal{L}(a) \mid a \in \mathcal{A}(\Pi) \setminus J\}, \text{ and} \\ Q_U &= \{\mathcal{L}(r) \mid r \in \mathcal{R}(\Pi), n(r) \cap I = \emptyset, J \notin \text{Mods}(r^+)\}. \end{aligned}$$

We can now define our dynamic programming algorithm for ASP:

Definition 8. The function $g : CW_k \rightarrow 2^{\mathcal{P}_k}$ is recursively defined along the structure of k -expressions as follows.

- $g(i(v)) = \left\{ ((\{i\}, \emptyset, \emptyset), \{(\emptyset, \{i\}, \emptyset)\}), ((\emptyset, \{i\}, \emptyset), \emptyset) \right\}$ if v is at atom node
- $g(i(v)) = \left\{ ((\emptyset, \emptyset, \{i\}), \emptyset) \right\}$ if v is a rule node

- $g(\sigma_1 \oplus \sigma_2) = \{(Q_1 \oplus Q_2, R_{Q_1, Q_2, \Gamma_1, \Gamma_2} \mid (Q_i, \Gamma_i) \in g(\sigma_i))\}$, where $R_{Q_1, Q_2, \Gamma_1, \Gamma_2} = \{S_1 \oplus S_2 \mid S_i \in \Gamma_i\} \cup \{Q_1 \oplus S \mid S \in \Gamma_2\} \cup \{S \oplus Q_2 \mid S \in \Gamma_1\}$
- $g(\rho_{i \rightarrow j}(\sigma)) = \{((Q^{i \rightarrow j}, \{R^{i \rightarrow j} \mid R \in \Gamma\}) \mid (Q, \Gamma) \in g(\sigma))\}$
- $g(\eta_{i,j}^h(\sigma)) = \{(Q^{Q_T, i, j}, \{R^{R_T, i, j} \mid R \in \Gamma\}) \mid (Q, \Gamma) \in g(\sigma)\}$
- $g(\eta_{i,j}^p(\sigma)) = \{(Q^{Q_F, i, j}, \{R^{R_F, i, j} \mid R \in \Gamma\}) \mid (Q, \Gamma) \in g(\sigma)\}$
- $g(\eta_{i,j}^n(\sigma)) = \{(Q^{Q_T, i, j}, \{R^{Q_T, i, j} \mid R \in \Gamma\}) \mid (Q, \Gamma) \in g(\sigma)\}$

Note the use of Q_T in $R^{Q_T, i, j}$ in the definition of $g(\eta_{i,j}^n(\sigma))$: Whenever an interpretation I represented by Q sets an atom from the negative body of a rule r to true, the rule r has no counterpart in the reduct w.r.t. I , so, for each subset J of I , we remove r from the set of rules whose counterpart in the reduct is not yet satisfied by J .

Example 4. Let Π be the program consisting of a single rule $\leftarrow \neg x$, which we denote by r , and let $\sigma = \eta_{1,2}^n(1(x) \oplus 2(r))$. Let (Q, Γ) be the k -pair in $g(1(x))$ with $Q = (\{1\}, \emptyset, \emptyset)$ and $\Gamma = \{(\emptyset, \{1\}, \emptyset)\}$. The k -triple Q represents the set of atoms $\{x\}$. Since this set has the proper subset \emptyset , there is a k -triple in Γ that indeed corresponds to this subset. Now let $(Q, \Gamma) = ((\emptyset, \{1\}, \emptyset), \emptyset)$ be the other k -pair in $g(1(x))$. Here Q represents the empty set of atoms, which has no proper subsets, hence Γ is empty. For the single k -pair $((\emptyset, \emptyset, \{2\}), \emptyset)$ in $g(2(r))$, the situation is similar. Next, at $g(1(x) \oplus 2(r))$, we combine every k -pair (Q_1, Γ_1) from $g(1(x))$ with every k -pair (Q_2, Γ_2) from $g(2(r))$ to a new k -pair. For instance, consider $Q_1 = (\{1\}, \emptyset, \emptyset)$ and $\Gamma_1 = \{S\}$, where $S = (\emptyset, \{1\}, \emptyset)$, as well as $Q_2 = (\emptyset, \emptyset, \{2\})$ and $\Gamma_2 = \emptyset$. By definition of g , we obtain a new k -pair (Q, Γ) , where $Q = Q_1 \oplus Q_2 = (\{1\}, \emptyset, \{2\})$, and Γ contains the single element $Q_2 \oplus S = (\emptyset, \{1\}, \{2\})$. Recall that the purpose of Q is to represent sets of atoms I , and each element of Γ shall represent proper subsets of I ; in this case, Q represents $\{x\}$, and the element $Q_2 \oplus S$ in Γ represents the proper subset \emptyset . Next, at $g(\sigma)$ we introduce a negative edge from x to r . From the k -pair $(Q, \{S\})$ in $g(1(x) \oplus 2(r))$, where $Q = (\{1\}, \emptyset, \{2\})$ and $S = (\emptyset, \{1\}, \{2\})$, we obtain the k -pair $(Q', \{S'\})$ in $g(\sigma)$, where $Q' = Q^{Q_T, i, j} = (\{1\}, \emptyset, \emptyset)$ (i.e., the label 2 from Q_U has disappeared) and $S' = S^{Q_T, i, j} = (\emptyset, \{1\}, \emptyset)$. Here 2 has disappeared from S_U because the reduct w.r.t. all sets of atoms represented by Q' no longer contains any rule labeled with 2. Note that the classical model $\{x\}$ represented by Q' is no answer set even though $Q'_U = \emptyset$. The reason is that S' witnesses (by $S'_U = \emptyset$) that $\emptyset \in \text{Mods}(\Pi^{\{x\}})$.

We now prove correctness of the algorithm from Definition 8.

Lemma 4. Let Π be a program and θ be a k -expression for $\text{SINC}(\Pi)$. For every set $I \subseteq \mathcal{A}(\Pi)$ there is a k -pair $(Q, \Gamma) \in g(\theta)$ such that (i) I is a Π -interpretation of Q and (ii) for every set $J \subset I$ there is a k -triple $R \in \Gamma$ such that J is a Π^I -interpretation of R . Moreover, for every k -pair $(Q, \Gamma) \in g(\theta)$ there is a set $I \subseteq \mathcal{A}(\Pi)$ such that (i') I is a Π -interpretation of Q and (ii') for each k -triple $R \in \Gamma$, there is a set $J \subset I$ such that J is a Π^I -interpretation of R .

Proof. Observe that for each (Q, Γ) in $g(\theta)$ it holds that $Q \in f(\theta)$, and for each Q in $f(\theta)$ there is some (Q, Γ) in $g(\theta)$. Hence we can apply the same arguments as in the proof of Lemma 3 for (i) and (i'). In addition, similar arguments can be used within each of the distinguished cases for (ii) and (ii'). We use induction on the structure of a k -expression θ defining Π . Let σ be a subexpression of θ , let Π_σ denote the program defined by σ , and let $I \subseteq \mathcal{A}(\Pi_\sigma)$.

If $\sigma = i(r)$, for $r \in \mathcal{R}(\Pi)$, then $\mathcal{A}(\Pi_\sigma) = \emptyset$, so $I = \emptyset$. As in the proof of Lemma 3, we can show that I is a Π_σ -interpretation of the k -triple $Q = (\emptyset, \emptyset, \{i\})$. Since I has no proper subsets, condition (ii) holds trivially for the k -pair (Q, \emptyset) in $g(\sigma)$.

If $\sigma = i(a)$, for $a \in \mathcal{A}(\Pi)$, then $\mathcal{A}(\Pi_\sigma) = \{a\}$ and $\mathcal{R}(\Pi_\sigma) = \emptyset$. If $I = \emptyset$, then I is a Π_σ -interpretation of the k -triple $Q = (\emptyset, \{i\}, \emptyset)$ and (ii) holds trivially for the k -pair (Q, \emptyset) in $g(\sigma)$. Otherwise $I = \{a\}$ holds. Let (Q, Γ) be the k -pair in $g(\sigma)$ for which $Q = (\{i\}, \emptyset, \emptyset)$ and $\Gamma = \{(\emptyset, \{i\}, \emptyset)\}$ hold. Clearly I is a Π_σ -interpretation of Q . The only proper subset of I is \emptyset , which is a Π_σ^I -interpretation of the only element of Γ .

If $\sigma = \sigma_1 \oplus \sigma_2$, let $\Pi_i = \Pi_{\sigma_i}$ and $I_i = I \cap \mathcal{A}(\Pi_i)$, for any $i \in \{1, 2\}$. By definition of Π_i , it holds that $\mathcal{A}(\Pi) = \mathcal{A}(\Pi_1) \cup \mathcal{A}(\Pi_2)$, $\mathcal{R}(\Pi) = \mathcal{R}(\Pi_1) \cup \mathcal{R}(\Pi_2)$ and $I = I_1 \cup I_2$. Let $J \subset I$ and, for $i \in \{1, 2\}$, let

$J_i = J \cap \mathcal{A}(\Pi_i)$. Observe that $J_1 \subseteq I_1$ and $J_2 \subseteq I_2$, and at least one inclusion is proper. We distinguish three cases:

- If $J_1 \subset I_1$ and $J_2 \subset I_2$, then, by induction hypothesis, for any $i \in \{1, 2\}$ there is a k -pair (Q_i, Γ_i) in $g(\sigma_i)$ such that I_i is a Π_i -interpretation of Q_i and there is a k -triple $R_i \in \Gamma_i$ such that J_i is a $\Pi_i^{I_i}$ -interpretation of R_i . By definition of g , there is a k -pair (Q, Γ) in $g(\sigma)$ such that $Q = Q_1 \oplus Q_2$ and there is a k -triple R in Γ such that $R = R_1 \oplus R_2$. We can easily check that I is a Π_σ -interpretation of Q and that J is a Π_σ^I -interpretation of R .
- If $J_1 \subset I_1$ and $J_2 = I_2$, then, by induction hypothesis, there is a k -pair (Q_1, Γ_1) in $g(\sigma_1)$ such that I_1 is a Π_1 -interpretation of Q_1 and there is a k -triple $R_1 \in \Gamma_1$ such that J_1 is a $\Pi_1^{I_1}$ -interpretation of R_1 . Also, there is a k -pair (Q_2, Γ_2) in $g(\sigma_2)$ such that I_2 is a Π_2 -interpretation of Q_2 . By definition of g , there is a k -pair (Q, Γ) in $g(\sigma)$ such that $Q = Q_1 \oplus Q_2$ and there is a k -triple R in Γ such that $R = R_1 \oplus Q_2$. We can easily check that I is a Π_σ -interpretation of Q and that J is a Π_σ^I -interpretation of R .
- The case $J_1 = I_1$, $J_2 \subset I_2$ is symmetric.

We omit the cases $\sigma = \rho_{i \rightarrow j}(\sigma')$, $\sigma = \eta_{i,j}^h(\sigma')$ and $\sigma = \eta_{i,j}^p(\sigma')$, as they do not offer much additional insight, given the proof of Lemma 3 and the following case.

If $\sigma = \eta_{i,j}^n(\sigma')$, then there is again a k -pair (Q', Γ') in $g(\sigma')$ such that (i) I is a $\Pi_{\sigma'}$ -interpretation of Q' and (ii) for each $J \subset I$ there is a k -triple R in Γ' such that J is a $\Pi_{\sigma'}^I$ -interpretation of R . Let (Q, Γ) be the k -pair in $g(\sigma)$ for which $Q = Q'^{Q_T, i, j}$ and $\Gamma = \{R^{Q_T, i, j} \mid R \in \Gamma'\}$ hold. As before, I is a Π_σ -interpretation of Q . Let $J \subset I$, let R' be the k -triple in Γ' such that J is a $\Pi_{\sigma'}^I$ -interpretation of R' , and let $R = R'^{Q_T, i, j}$. As before, J satisfies the first two conditions for being a Π_σ^I -interpretation of R . It remains to check the third condition.

For all labels except j , we proceed as before. We now check that $j \in R_U$ if and only if there is a rule $r \in \mathcal{R}(\Pi_\sigma)$ such that $\mathcal{L}_\sigma(r) = j$, $n(r) \cap I = \emptyset$ and $J \notin \text{Mods}(r^+)$. First suppose toward a contradiction that $j \in R_U$ while $J \in \text{Mods}(r^+)$ for each $r \in \mathcal{R}(\Pi_\sigma)$ such that $\mathcal{L}_\sigma(r) = j$ and $n(r) \cap I = \emptyset$. Since $R_U \subseteq R'_U$, also $j \in R'_U$ and by induction hypothesis there is a rule $r' \in \mathcal{R}(\Pi_{\sigma'})$ such that $\mathcal{L}_{\sigma'}(r') = j$, $n(r') \cap I = \emptyset$ and J is not a model of r'^+ . There is a corresponding rule $r \in \mathcal{R}(\Pi_\sigma)$, for which $\mathcal{L}_\sigma(r) = j$, $h(r') = h(r)$, $p(r') = p(r)$ and $n(r') \subseteq n(r)$ hold. Since J is a model of r^+ but not of r'^+ (and these rules are identical), there is an $a \in n(r) \cap I$ with $\mathcal{L}_\sigma(a) = i$. From $a \in I$ it follows by induction hypothesis that $i \in Q'_T$, but this leads to the contradiction $j \notin R_U$.

Finally, suppose toward a contradiction that $j \notin R_U$ and there is a rule $r \in \mathcal{R}(\Pi_\sigma)$ such that $\mathcal{L}_\sigma(r) = j$, $n(r) \cap I = \emptyset$ and $J \notin \text{Mods}(r^+)$. Let r' be the rule corresponding to r in $\Pi_{\sigma'}$. For this rule it holds that $\mathcal{L}_{\sigma'}(r') = j$ and $n(r') \subseteq n(r)$, so $n(r') \cap I = \emptyset$. The set J does not satisfy r'^+ either, since $h(r'^+) = h(r^+)$ and $p(r'^+) = p(r^+)$. By induction hypothesis, this entails $j \in R'_U$. Due to $j \notin R_U$, it holds that $i \in Q'_T$, so there is an atom $a \in I$ such that $\mathcal{L}_{\sigma'} = \mathcal{L}_\sigma(a) = i$. Due to the new edge from a to r , $a \in n(r)$ holds, which contradicts $n(r) \cap I = \emptyset$.

The other direction is similar. □

Hence we get an FPT result for answer-set semantics:

Theorem 4. *Let k be a constant and Π be a program. Given a k -expression for the signed incidence graph of Π , we can decide in linear time whether Π has an answer set.*

Proof. Let k be a constant, Π be a program and σ be a k -expression of $\text{SINC}(\Pi)$. Using the same ideas as for Theorem 3, we can easily show that there is an answer set of Π if and only if there is a k -pair (Q, Γ) in $g(\sigma)$ such that $Q_U = \emptyset$ and $R_U \neq \emptyset$ for every $R \in \Gamma$. Again, it is easy to see that $g(\sigma)$ can be computed in linear time. □

4.2 The Role of Signs for Results on Clique-Width

We have shown that ASP parameterized by the clique-width of the signed incidence graph is FPT. Because the clique-width of the (unsigned) incidence graph is usually smaller than (and always at most as high as) the clique-width of the signed incidence graph (Proposition 2), an FPT result w.r.t. the clique-width of the (unsigned)

incidence graph would be a significantly stronger result. It is therefore natural to ask whether ASP is already FPT w.r.t the clique-width of the unsigned incidence graph. A similar situation is known for the satisfiability problem of propositional formulas (SAT), which was first shown in [27] to be FPT parameterized by the clique-width of the signed incidence graph, and the authors conjectured that the same should hold already for the unsigned version. Surprisingly, this turned out not to be the case [46]. In comparison to SAT, the situation for ASP is similar but slightly more involved. Whereas there are only two potential signs for SAT (signaling whether a variable occurs positively or negatively in a clause), ASP has three signs (h, p, n) . So how many signs are necessary to obtain tractability for ASP? To settle this question, let $\text{SINC}_L(\Pi)$, for $L \subseteq \{h, p, n\}$, be the (“semi-signed”) incidence graph obtained from $\text{SINC}(\Pi)$ by joining all labels in L , i.e., every label in L is renamed to a new label, which we denote by α . We will show below that joining any set L of labels other than $\{h, n\}$ leads to intractability for ASP parameterized by the clique-width of $\text{SINC}_L(\Pi)$. Together with our tractability result w.r.t. the clique-width of $\text{SINC}(\Pi)$ (Theorem 4), this provides an almost complete picture of the complexity of ASP parameterized by clique-width. We leave it as an open question whether ASP parameterized by the clique-width of $\text{SINC}_{\{h, n\}}(\Pi)$ is FPT.

Theorem 5. *Let $L \subseteq \{h, p, n\}$ with $|L| > 1$ and $L \neq \{h, n\}$, then ASP is $W[1]$ -hard parameterized by the clique-width of $\text{SINC}_L(\Pi)$.*

Proof. We show the result by a parameterized reduction from the $W[1]$ -complete problem PARTITIONED CLIQUE [48].

Instance: A k -partite graph $G = (V, E)$ with partition V_1, \dots, V_k where $|V_i| = |V_j|$ for every i, j with $1 \leq i, j \leq k$.

Parameter: The integer k .

Question: Does G have a clique of size k ?

Recall that a k -partite graph is a graph whose vertex set can be partitioned into k sets such that there are no edges between vertices contained in the same set. To prove the theorem it is sufficient to show that the result holds for L being any combination of two labels other than $\{h, n\}$. In other words, it suffices to show the result for $L = \{h, p\}$ and $L = \{p, n\}$. Because the reduction for the case that $L = \{h, p\}$ is very similar to the reduction from PARTITIONED CLIQUE to SAT given in [46, Corollary 1], we omit its proof here and only give the proof for the case that $L = \{p, n\}$. Let $L = \{p, n\}$ and G, k, V_1, \dots, V_k be as in the definition of PARTITIONED CLIQUE and assume that the vertices of G are labeled such that v_i^j is the i -th vertex contained in V_j . We will construct a program Π in polynomial-time such that G has a clique of size k if and only if Π has an answer set, and the clique-width of $\text{SINC}_L(\Pi)$ is at most $k' = 2k + k^2$.

The program Π contains one atom v_i^j for every vertex v_i^j of G , and the following rules:

(R1) For every j with $1 \leq j \leq k$, the rule: $v_1^j \vee \dots \vee v_n^j \leftarrow$.

(R2) The rule:

$$\begin{aligned} \leftarrow v_{i_1}^{j_1}, v_{i_2}^{j_2}, \neg v_1^{j_1}, \dots, \neg v_{i_1-1}^{j_1}, \neg v_{i_1+1}^{j_1}, \dots, \neg v_n^{j_1}, \\ \neg v_1^{j_2}, \dots, \neg v_{i_2-1}^{j_2}, \neg v_{i_2+1}^{j_2}, \dots, \neg v_n^{j_2} \end{aligned}$$

for every $\{v_{i_1}^{j_1}, v_{i_2}^{j_2}\} \notin E(G)$ with $1 \leq j_1, j_2 \leq k$ and $1 \leq i_1, i_2 \leq n$.

We first show that G has a clique of size k if and only if Π has an answer set. Toward showing the forward direction, let C be a clique of size k of G . We claim that C is also an answer set of Π and first show that C is indeed a model of Π . Because C contains exactly one vertex from every V_i , all rules of type (R1) are satisfied by C . Moreover, the same holds for all rules of type (R2), because there is no pair $u, v \in C$ with $\{u, v\} \notin E(G)$ and hence the body of every such rule is always falsified. This shows that C is a model of Π . Finally, because all the rules of type (R1) are also contained in the reduct Π^C of Π , we obtain that C is an answer set of Π .

Toward showing the reverse direction, let C be an answer set of Π . We claim that C is also a clique of G of size k and first show that C contains exactly one variable from every V_i . Because of the rules of type (R1) (which will also remain in the reduct Π^C), it holds that C contains at least one variable corresponding to a vertex of V_i for every i with $1 \leq i \leq k$. Assume for a contradiction that C contains more than one variable from some V_i . Then for every j with $j \neq i$, C has to contain at least three variables from $V_i \cup V_j$. Consequently, every rule of type (R2) corresponding to a non-edge of G incident to a vertex in V_i does not appear in the reduct Π^C of Π ,

which shows that C is not an answer set (minimal model) of Π^C . This shows that C contains exactly one variable from every V_i . Now, suppose for a contradiction that C is not a clique of G of size k . Then there are u and w in C with $u \in V_i$ and $w \in V_j$ such that $\{u, w\} \notin E(G)$. Hence, there is a rule of type (R2) (corresponding to the non-edge $\{u, v\}$), which is violated by C , a contradiction to our assumption that C is model of Π .

It remains to show that the clique-width of $\text{SINC}_L(\Pi)$ is at most $k' = 2k + k^2$. We show this by providing a k' -expression for $\text{SINC}_L(\Pi)$. We start by giving the terms that introduce the vertices of $\text{SINC}_L(\Pi)$: (1) We introduce every atom vertex v_i^j of $\text{SINC}_L(\Pi)$ with the term $j(v_i^j)$. (2) For every rule vertex r of the form (R1) corresponding to a rule $v_1^j \vee \dots \vee v_n^j \leftarrow$, we introduce the term $l(r)$ where $l = k + j$. (3) For every rule vertex r of the form (R2) corresponding to a non-edge between V_i and V_j with $1 \leq i < j \leq k$, we introduce the term $l(r)$, where $l = 2k + k(i - 1) + j$. We then combine all these terms using the disjoint union operator \oplus . Now it only remains to show how the edges between the rule and the atom vertices of $\text{SINC}_L(\Pi)$ are added: First, for every j with $1 \leq j \leq k$, we add the edges between the rule vertices of the form (R1) and the atom vertices contained in those rules using the operator $\eta_{j, k+j}^\ell$, where $\ell = h$. Second, for every i and j with $1 \leq i < j \leq k$, we add the edges between the rule vertices of the form (R2) and the atom vertices contained in those rules using the operators $\eta_{i, 2k+k(i-1)+j}^\alpha$ and $\eta_{j, 2k+k(i-1)+j}^\alpha$.

□

5 Conclusion

In this paper, we have contributed to the parameterized complexity analysis of ASP. We first gave some negative observations showing that most directed width measures (applied to the dependency graph or incidence graph of a program) do not lead to FPT results. On the other hand, we turned a theoretical tractability result (which implicitly follows from previous work [31]) for the parameter clique-width (applied to the signed incidence graph of a program) into a novel dynamic programming algorithm. The algorithm is applicable to arbitrary programs, whenever a defining k -expression is given. The algorithm is expected to run efficiently in particular for small k , i.e., programs for which the signed incidence graph has low clique-width.

Future work includes solving the remaining question whether ASP parameterized by the clique-width of $\text{SINC}_{\{h, n\}}(\Pi)$ is FPT or not. Another open question is whether ASP parameterized by the clique-width of the unsigned incidence graph is in the class XP (as is the case for SAT [51]).

Acknowledgments. This work was supported by the Austrian Science Fund (FWF) projects P25518 and Y698.

References

- [1] G. Bagan, A. Bonifati, and B. Groz. A trichotomy for regular simple path queries on graphs. In *Proc. PODS*, pages 261–272. ACM, 2013.
- [2] M. Balduccini, M. Gelfond, and M. Nogueira. Answer set based design of knowledge systems. *Ann. Math. Artif. Intell.*, 47(1-2):183–219, 2006.
- [3] J. Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2): 161–172, 2006.
- [4] R. Ben-Eliyahu. A hierarchy of tractable subsets for computing stable models. *J. Artif. Intell. Res. (JAIR)*, 5:27–52, 1996. doi: 10.1613/jair.223. URL <http://dx.doi.org/10.1613/jair.223>.
- [5] R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Ann. Math. Artif. Intell.*, 12(1-2):53–87, 1994. doi: 10.1007/BF01530761. URL <http://dx.doi.org/10.1007/BF01530761>.
- [6] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, and J. Obdržálek. The dag-width of directed graphs. *Journal of Combinatorial Theory, Series B*, 102(4):900–923, 2012.
- [7] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.
- [8] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, front-size, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, 1995.
- [9] M. Bojańczyk, C. Dittmann, and S. Kreutzer. Decomposition theorems and model-checking for the modal μ -calculus. In *Proc. CLS/LICS*, pages 17:1–17:10. ACM, 2014.
- [10] G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Comm. ACM*, 54(12): 92–103, 2011.
- [11] R. S. Cohen. Transition graphs and the star height problem. In *Proc. of the 9th Annual Symposium on Switching and Automata Theory*, pages 383–394. IEEE Computer Society, 1968.
- [12] D. G. Corneil and U. Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005. doi: 10.1137/S0097539701385351. URL <http://dx.doi.org/10.1137/S0097539701385351>.
- [13] B. Courcelle. Recognizability and second-order definability for sets of finite graphs. Technical Report I-8634, Université de Bordeaux, 1987.
- [14] B. Courcelle. Graph rewriting: an algebraic and logic approach. In *Handbook of theoretical computer science, Vol. B*, pages 193–242. Elsevier Science Publishers, North-Holland, Amsterdam, 1990.
- [15] B. Courcelle and S. Olariu. Upper bounds to the clique-width of graphs. *Discr. Appl. Math.*, 101(1-3): 77–114, 2000.
- [16] B. Courcelle, J. Engelfriet, and G. Rozenberg. Context-free handle-rewriting hypergraph grammars. In *Proc. Graph-Grammars*, volume 532 of *LNCS*, pages 253–268, 1991.
- [17] B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *JCSS*, 46(2):218–270, 1993.
- [18] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.
- [19] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015. ISBN 978-3-319-21274-6. doi: 10.1007/978-3-319-21275-3.

- [20] M. Dehmer and F. Emmert-Streib, editors. *Quantitative Graph Theory*, chapter Width-Measures for Directed Graphs and Algorithmic Applications. CRC Press, 2014.
- [21] R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- [22] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- [23] W. Dvorák, S. Szeider, and S. Woltran. Reasoning in argumentation frameworks of bounded clique-width. In *Proc. COMMA*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 219–230. IOS Press, 2010. ISBN 978-1-60750-618-8. URL <http://dx.doi.org/10.3233/978-1-60750-619-5-219>.
- [24] T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.*, 15(3–4):289–323, 1995.
- [25] M. R. Fellows, F. A. Rosamond, U. Rotics, and S. Szeider. Clique-width is NP-complete. *SIAM J. Discrete Math.*, 23(2):909–939, 2009.
- [26] J. Fichte and S. Szeider. Backdoors to tractable answer set programming. *Artif. Intell.*, 220:64–103, 2015.
- [27] E. Fischer, J. A. Makowsky, and E. R. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discr. Appl. Math.*, 156(4):511–529, 2008.
- [28] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012. doi: 10.2200/S00457ED1V01Y201211AIM019. URL <http://dx.doi.org/10.2200/S00457ED1V01Y201211AIM019>.
- [29] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.
- [30] A. C. Giannopoulou, P. Hunter, and D. M. Thilikos. LIFO-search: A min-max theorem and a searching game for cycle-rank and tree-depth. *Discrete Applied Mathematics*, 160(15):2089–2097, 2012.
- [31] G. Gottlob, R. Pichler, and F. Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artif. Intell.*, 174(1):105–132, 2010.
- [32] H. Gruber. Digraph complexity measures and applications in formal language theory. *Discrete Mathematics & Theoretical Computer Science*, 14(2):189–204, 2012.
- [33] C. Guziolowski, S. Videla, F. Eduati, S. Thiele, T. Cokelaer, A. Siegel, and J. Saez-Rodriguez. Exhaustively characterizing feasible logic models of a signaling network using answer set programming. *Bioinformatics*, 29(18):2320–2326, 2013. doi: 10.1093/bioinformatics/btt393. Erratum see *Bioinformatics* 30, 13, 1942.
- [34] M. Heule and S. Szeider. A SAT approach to clique-width. *ACM Trans. Comput. Log.*, 16(3):24, 2015. doi: 10.1145/2736696. URL <http://doi.acm.org/10.1145/2736696>.
- [35] P. Hunter and S. Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *TCS*, 399(3): 206–219, 2008.
- [36] M. Jakl, R. Pichler, and S. Woltran. Answer-set programming with bounded treewidth. In *Proc. IJCAI*, pages 816–822, 2009.
- [37] T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.
- [38] L. Kaiser, S. Kreutzer, R. Rabinovich, and S. Siebertz. Directed width measures and monotonicity of directed graph searching. *CoRR*, abs/1408.4745, 2014. URL <http://arxiv.org/abs/1408.4745>.

- [39] M. Kaminski, V. V. Lozin, and M. Milanic. Recent developments on graphs of bounded clique-width. *Discrete Applied Mathematics*, 157(12):2747–2761, 2009.
- [40] M. M. Kanté. The rank-width of directed graphs. *CoRR*, abs/0709.1433, 2007.
- [41] M. Lackner and A. Pfandler. Fixed-parameter algorithms for finding minimal models. In *Proc. KR*, pages 85–95. AAAI Press, 2012.
- [42] F. Lin and X. Zhao. On odd and even cycles in normal logic programs. In *Proc. AAAI*, pages 80–85. AAAI Press / The MIT Press, 2004.
- [43] Z. Lonc and M. Truszczyński. Fixed-parameter complexity of semantics for logic programs. *ACM Trans. Comput. Log.*, 4(1):91–119, 2003.
- [44] V. W. Marek and M. Truszczyński. Stable Models and an Alternative Logic Programming Paradigm. In *The Logic Programming Paradigm – A 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.
- [45] M. Morak, R. Pichler, S. Rümmele, and S. Woltran. A dynamic-programming based ASP-solver. In *Proc. JELIA’10*, pages 369–372, 2010. doi: 10.1007/978-3-642-15675-5_34. URL http://dx.doi.org/10.1007/978-3-642-15675-5_34.
- [46] S. Ordyniak, D. Paulusma, and S. Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. *TCS*, 481:85–99, 2013.
- [47] S. Oum and P. Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4): 514–528, 2006.
- [48] K. Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *JCSS*, 67(4):757–771, 2003.
- [49] F. Ricca, G. Grasso, M. Alviano, M. Manna, V. Lio, S. Iritano, and N. Leone. Team-building with answer set programming in the Gioia-Tauro seaport. *TPLP*, 12:361–381, 4 2012. ISSN 1475-3081. doi: 10.1017/S147106841100007X.
- [50] M. A. Safari. D-width: A more natural measure for directed tree width. In *Proc. MFCS*, volume 3618 of *LNCS*, pages 745–756. Springer, 2005. ISBN 3-540-28702-7.
- [51] F. Slivovsky and S. Szeider. Model counting for formulas of bounded clique-width. In *Proc. ISAAC*, volume 8283 of *LNCS*, pages 677–687. Springer, 2013.
- [52] T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In *Proc. PADL*, volume 1551 of *LNCS*, pages 305–319. Springer Verlag, 1998.
- [53] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proc. Theory of Computing*, pages 1–9. ACM, 1973.
- [54] M. Truszczyński. Trichotomy and dichotomy results on the complexity of reasoning with disjunctive logic programs. *TPLP*, 11(6):881–904, 2011. doi: 10.1017/S1471068410000463. URL <http://dx.doi.org/10.1017/S1471068410000463>.